

Table of Contents

This slide deck consists of slides used in 2 lecture videos in Week 3. Below is a list of shortcut hyperlinks for you to jump into specific sections.

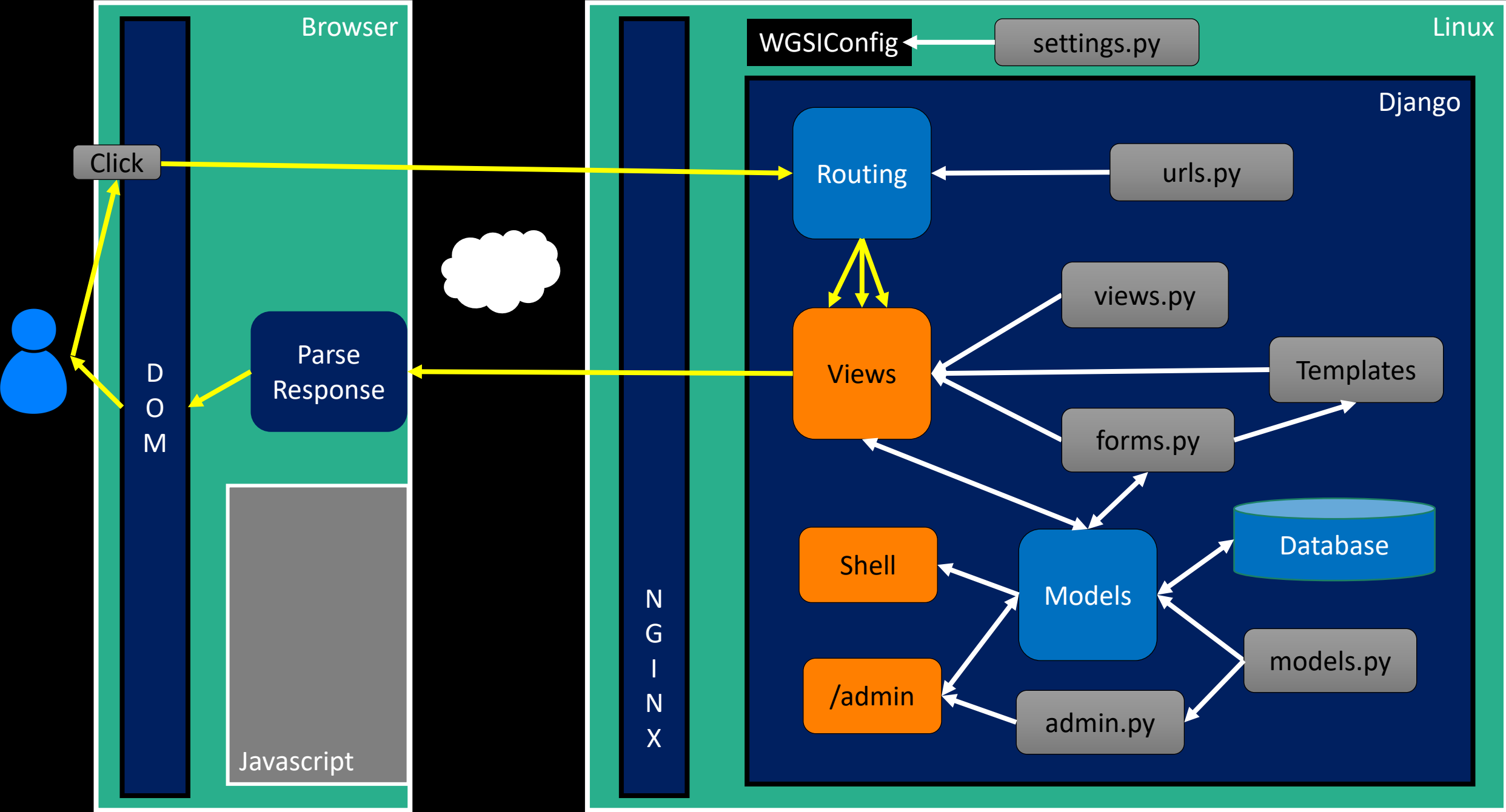
- (page 2) [Week 3: Using Django Forms Capabilities](#)
- (page 16) [Week 3: Data Validation with Django Forms](#)

Charles Severance
www.dj4e.com

Forms in Django

<https://samples.dj4e.com/form/>
<https://docs.djangoproject.com/en/3.0/topics/forms/>
<https://github.com/csev/dj4e-samples/tree/master/form>





Django's role in forms (DRY)

Handling forms is a complex business. Consider Django's admin, where numerous items of data of several different types may need to be prepared for display in a form, rendered as HTML, edited using a convenient interface, returned to the server, validated and cleaned up, and then saved or passed on for further processing. Django's form functionality can simplify and automate vast portions of this work, and can also do it more securely than most programmers would be able to do in code they wrote themselves.

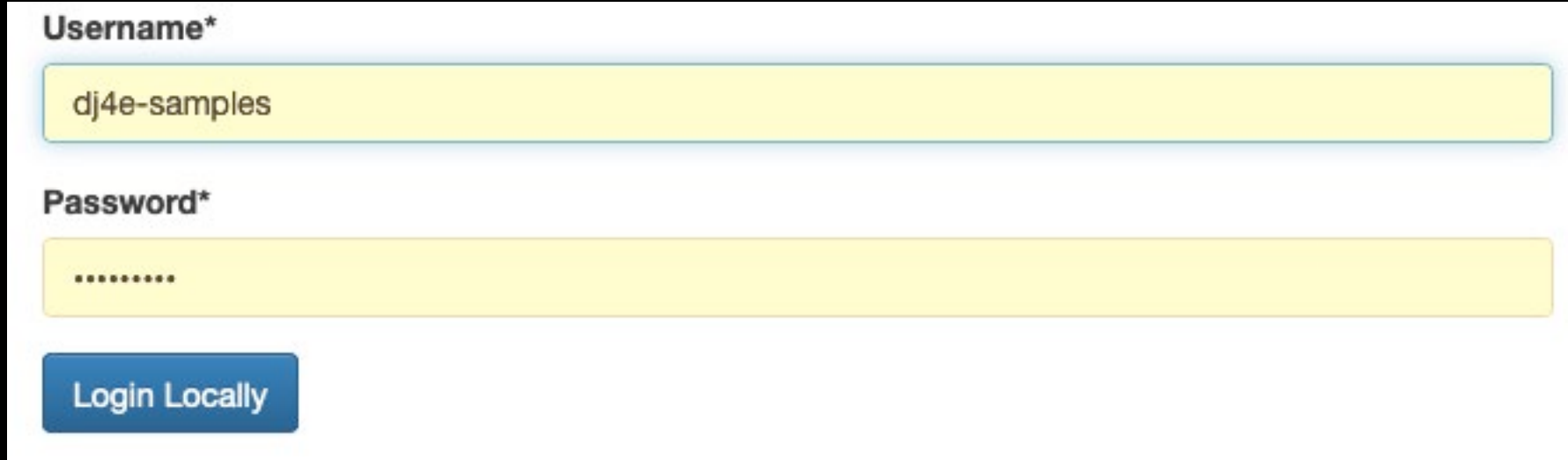
Django handles three distinct parts of the work involved in forms:

- preparing and restructuring data to make it ready for rendering
- creating HTML forms for the data
- receiving and processing submitted forms and data from the client

It is *possible* to write code that does all of this manually, but Django can take care of it all for you.

<https://docs.djangoproject.com/en/3.0/topics/forms/#django-s-role-in-forms>

It takes a lot of CSS to make forms pretty



Username*

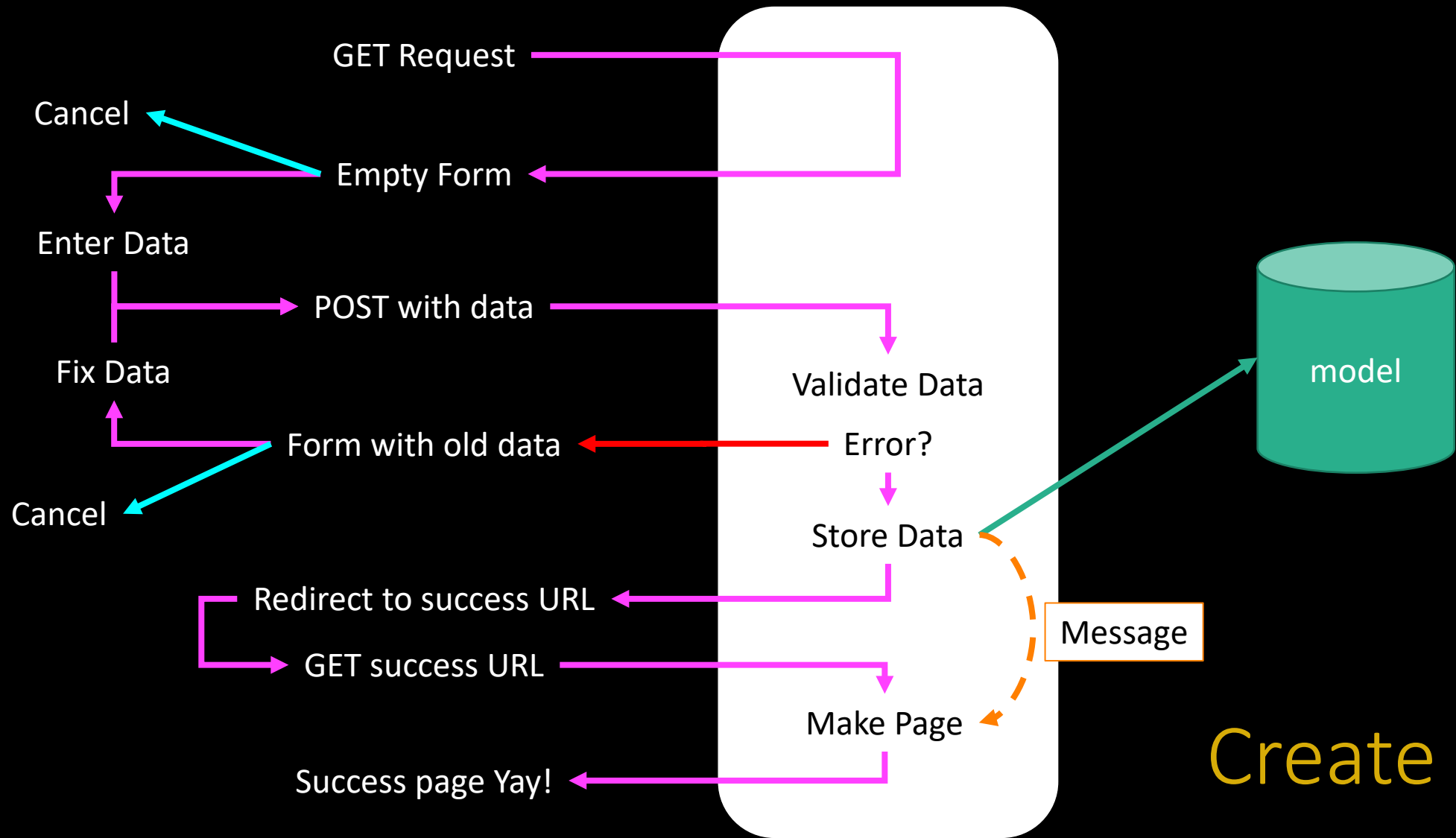
Password*

Login Locally

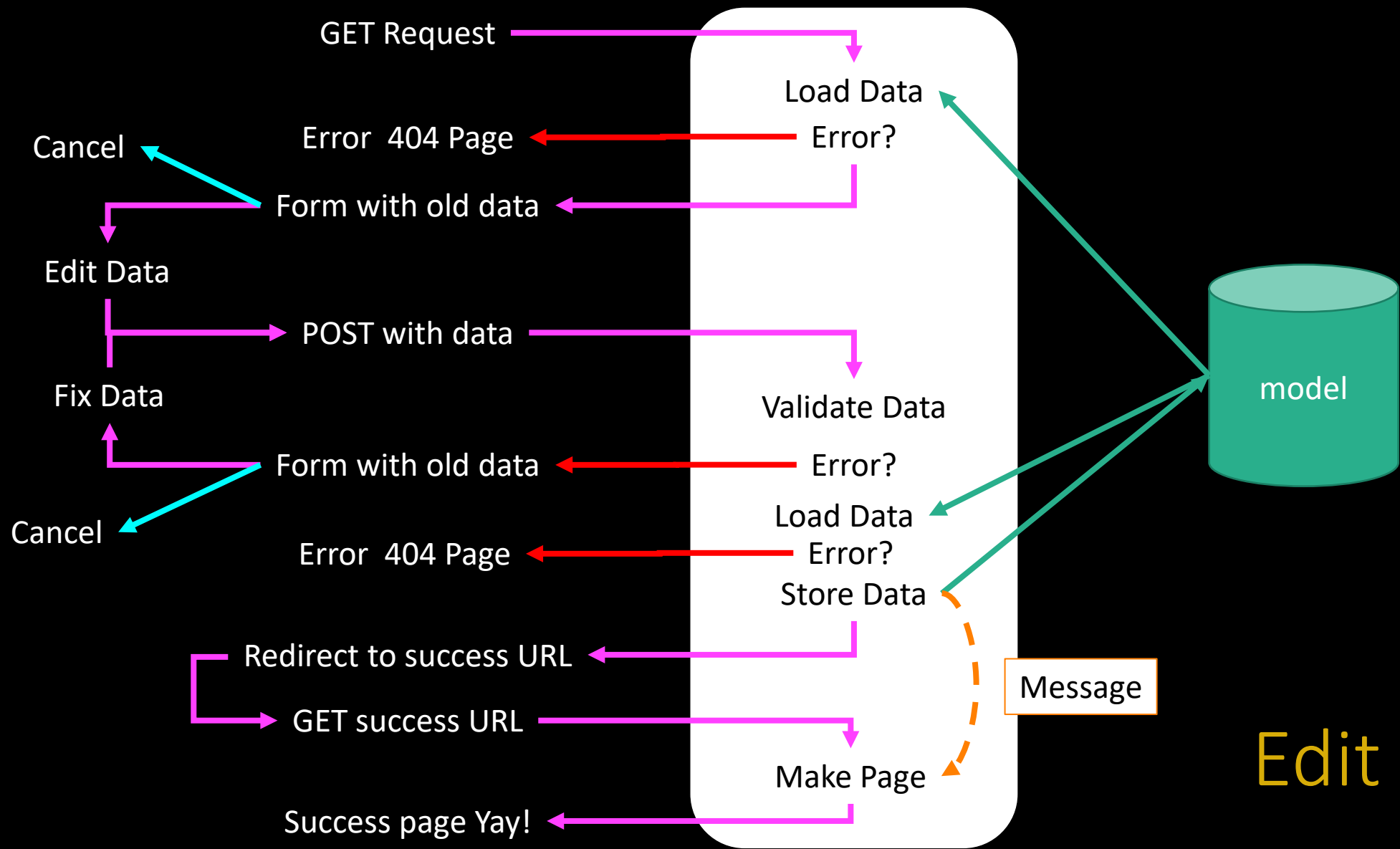
<https://samples.dj4e.com/accounts/login/>

Form Handling Flow is Actually Complex

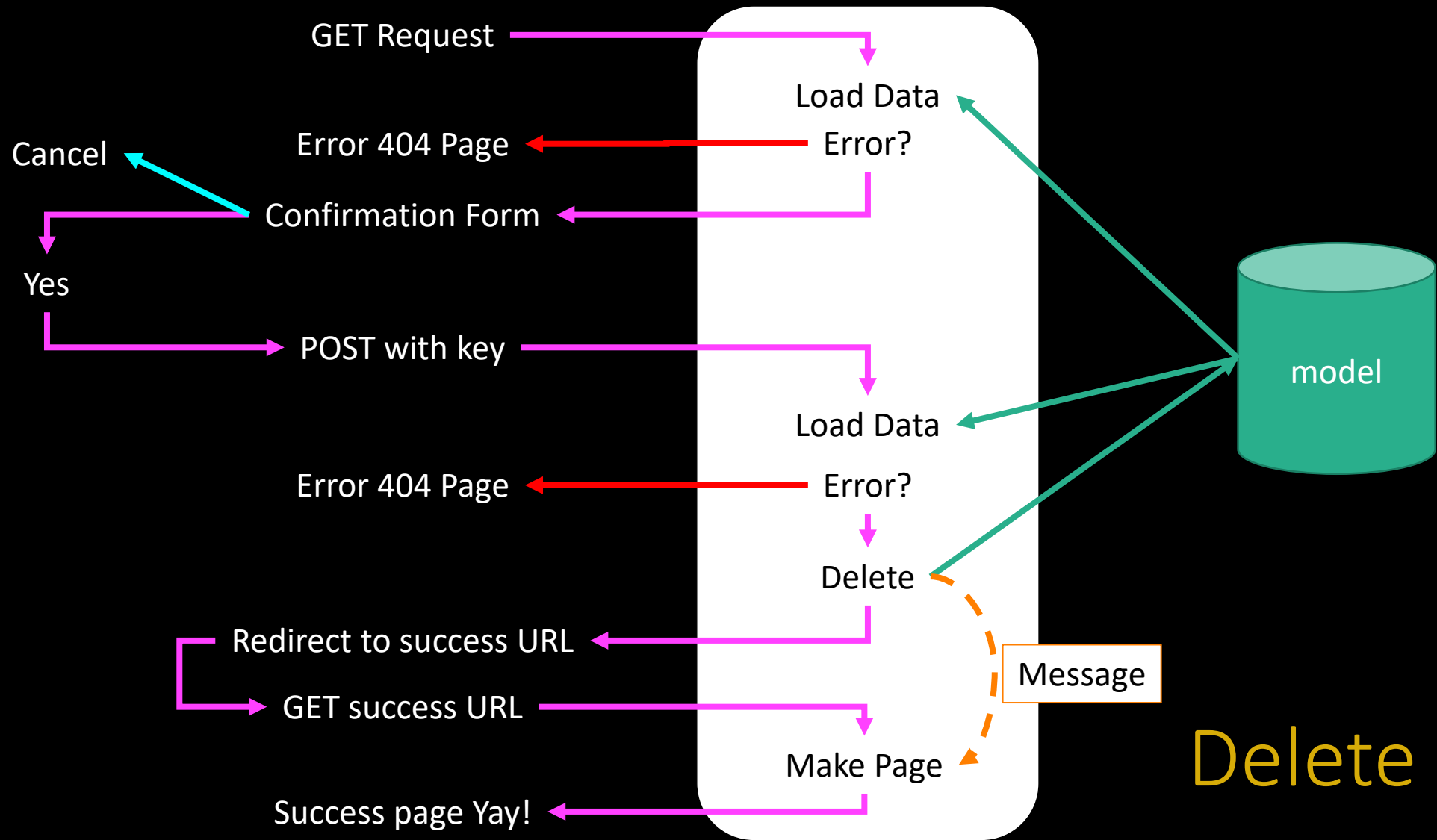
- Create
 - Produce empty form, check post data for validity, re-display form with errors if necessary, add the data to the database, and redirect the user to a success page with a success message
- Update
 - Load old data, form with old data, check post data for validity, re-display form with errors if necessary, update the data to the database, and redirect the user to a success page with a success message
- Delete
 - Load old data, produce confirmation page with a POST form, receive the post data, delete the record, and redirect the user to a success page with a success message



Create Form
Flow



Edit Form
Flow



Delete Form
Flow

Django forms act as "glue"

- Generate the necessary HTML to send to the browser
 - Allow for consistent look and feel across all the forms in an application
- Receive the POST data coming back from the browser
- Validate the incoming POST data and produce HTML for an error screen if necessary
- Move the data from the form into a model and then store it in the database automatically

<https://docs.djangoproject.com/en/3.0/topics/forms/#django-s-role-in-forms>

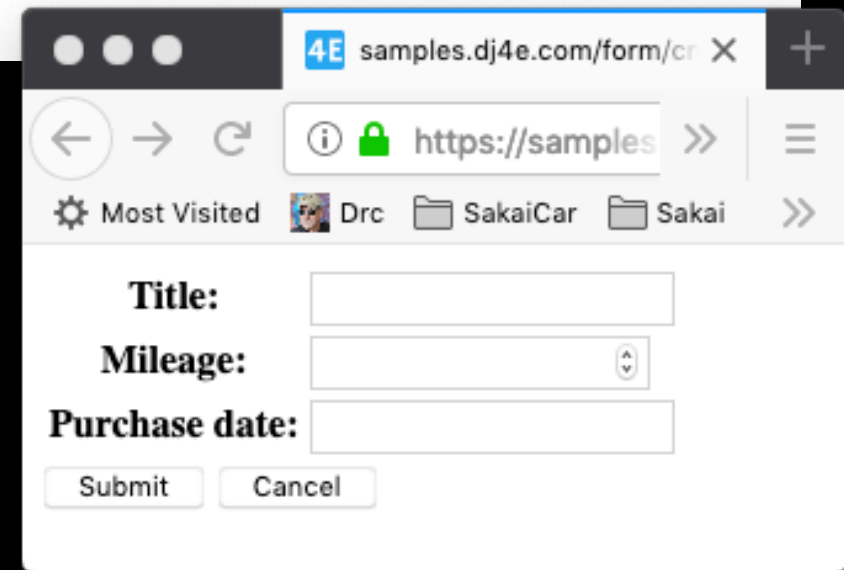
[dj4e-samples/form/forms.py](#)

```
from django import forms
from django.core.exceptions import ValidationError
from django.core import validators

class BasicForm(forms.Form):
    title = forms.CharField(validators=[
        validators.MinLengthValidator(2, "Please enter 2 or more characters")])
    mileage = forms.IntegerField()
    purchase_date = forms.DateField()
```

<https://samples.dj4e.com/form/create>

A simple form



4E samples.dj4e.com/form/create

← → ↻ ⓘ 🔒 https://samples >> ≡

⚙ Most Visited 🖼 Drc 📁 SakaiCar 📁 Sakai >>

Title:

Mileage:

Purchase date:

Dumping a form object

<https://samples.dj4e.com/form/example>

`dj4e-samples/form/forms.py`

```
class BasicForm(forms.Form):
    title = forms.CharField(validators=[
        validators.MinLengthValidator(2, "...")])
    mileage = forms.IntegerField()
    purchase_date = forms.DateField()
```

`dj4e-samples/form/views.py`

```
from form.forms import BasicForm

def example(request) :
    form = BasicForm()
    return HttpResponse(form.as_table())
```

```
<tr><th>
<label for="id_title">Title:</label></th>
<td><input type="text" name="title"
required id="id_title"></td></tr>
<tr><th>
<label for="id_mileage">Mileage:</label>
</th><td>
<input type="number" name="mileage" required
id="id_mileage">
</td></tr>
<tr><th>
<label for="id_purchase_date">
Purchase date:</label>
</th><td>
<input type="text" name="purchase_date"
required id="id_purchase_date">
</td></tr>
```

A form in a template

`dj4e-samples/form/templates/form/form.html`

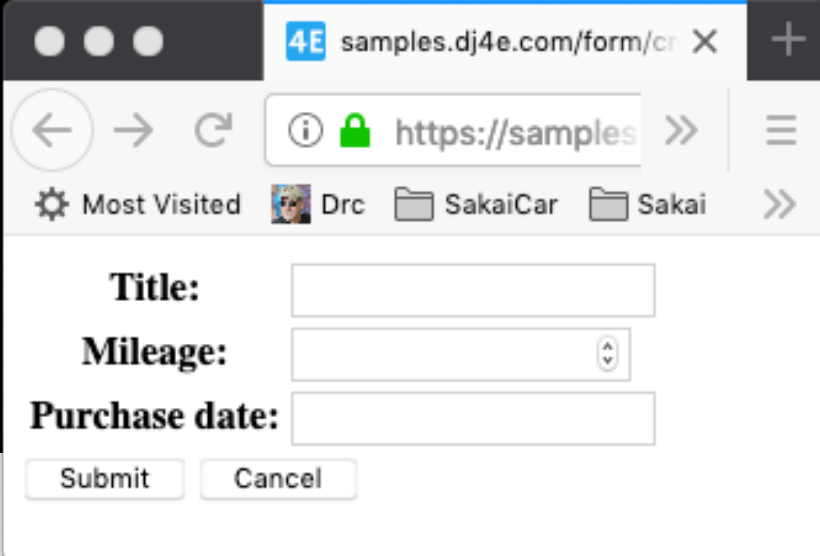
```
<p>
  <form action="" method="post">
    {% csrf_token %}
    <table>
      {{ form.as_table }}
    </table>
    <input type="submit" value="Submit">
    <input type="submit"
      onclick="window.location='{% url 'form:main' %}' ; return false;"
      value="Cancel">
  </form>
</p>
```

A form in a template

<https://samples.dj4e.com/form/create>

`dj4e-samples/form/views.py`

```
class SimpleCreate(DumpPostView):  
    def get(self, request) :  
        form = BasicForm()  
        ctx = {'form' : form}  
        return render(request, 'form/form.html', ctx)
```



The screenshot shows a web browser window with the address bar displaying `https://samples.dj4e.com/form/create`. The browser's address bar also shows a lock icon and a menu icon. Below the address bar, there is a navigation bar with a gear icon labeled 'Most Visited', a profile icon labeled 'Drc', and two folder icons labeled 'SakaiCar' and 'Sakai'. The main content area of the browser displays a form with three input fields: 'Title:', 'Mileage:', and 'Purchase date:'. The 'Mileage:' field has a small up/down arrow icon on its right side. Below the input fields are two buttons: 'Submit' and 'Cancel'.

Pulling existing data into a form

`https://samples.dj4e.com/form/update`

`dj4e-samples/form/views.py`

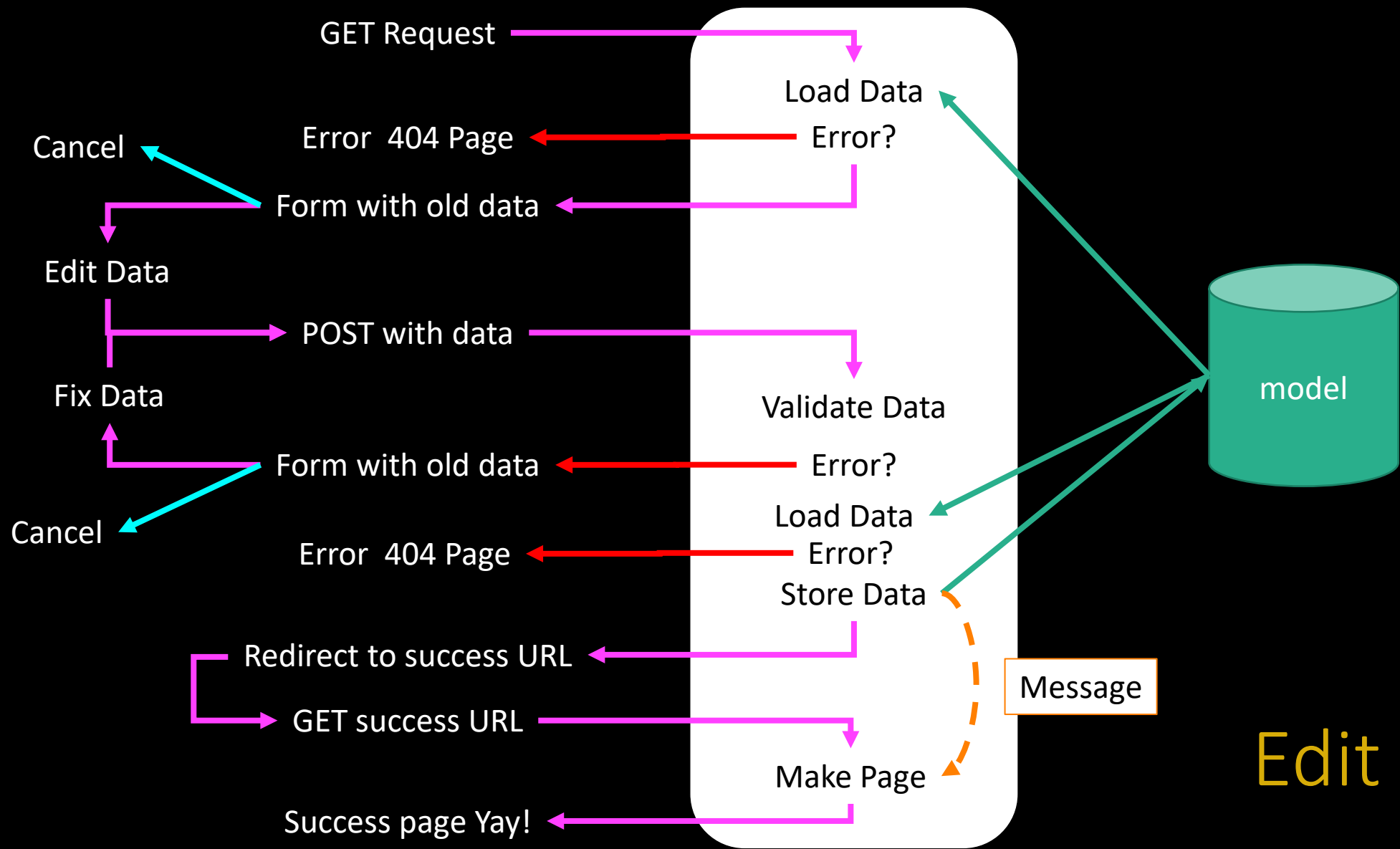
```
class SimpleUpdate(DumpPostView):
    def get(self, request):
        old_data = {
            'title': 'SakaiCar',
            'mileage' : 42,
            'purchase_date': '2018-08-14'
        }
        form = BasicForm(old_data)
        ctx = {'form' : form}
        return render(request, 'form/form.html', ctx)
```

The screenshot shows a web browser window with the address bar displaying 'https://samples.dj4e.com/form/update'. The browser's tab bar shows a single tab titled '4E samples.dj4e.com/form/up'. The browser's address bar also shows a green lock icon and a menu icon. Below the address bar, there is a 'Most Visited' section with three items: 'Drc', 'SakaiCar', and 'Sakai'. The main content area of the browser displays a form with the following fields:

- Title:** A text input field containing 'SakaiCar'.
- Mileage:** A text input field containing '42' with a small up/down arrow icon on the right.
- Purchase date:** A text input field containing '2018-08-14'.

At the bottom of the form, there are two buttons: 'Submit' and 'Cancel'.

Data Validation in FORMS

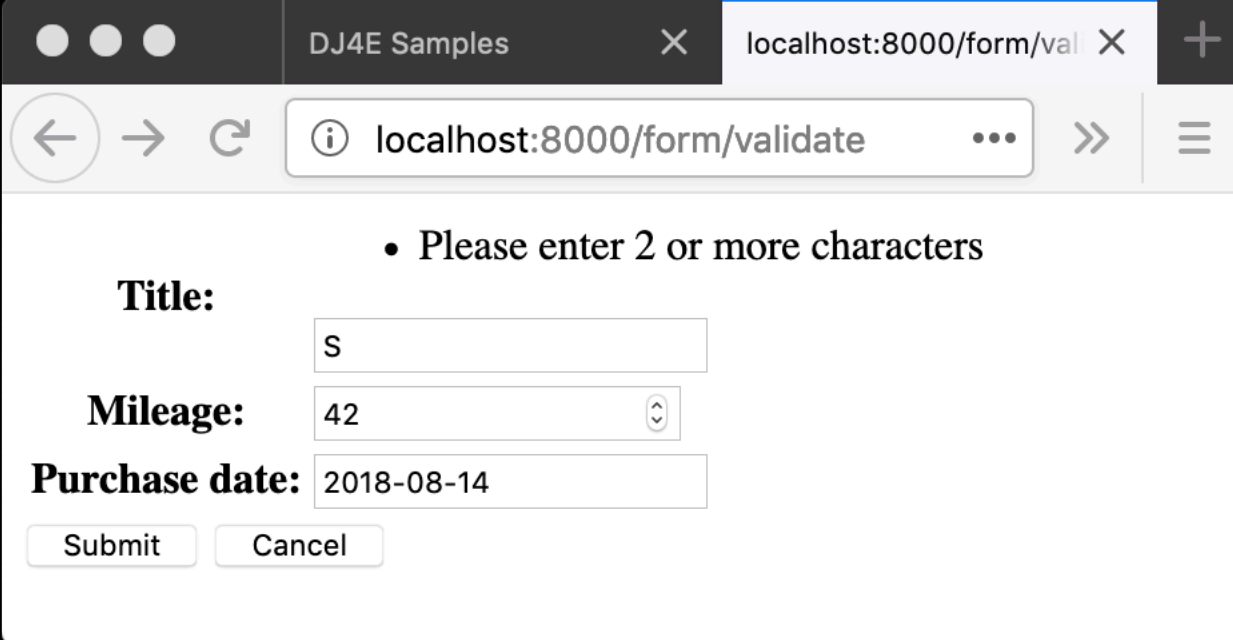


Edit Form
Flow

Form Data Errors

- Sometimes there are validation rules when you are filling out a form.
- When you submit the form, the view code checks the data to see if there are errors
- If there are errors, data is not saved and the user is notified and usually given a chance to edit and resubmit

<https://samples.dj4e.com/form/validate>



The screenshot shows a web browser window with two tabs: "DJ4E Samples" and "localhost:8000/form/val". The address bar displays "localhost:8000/form/validate". The page content includes a validation error message: "• Please enter 2 or more characters". Below this, there are three form fields: "Title:" with the value "S", "Mileage:" with the value "42", and "Purchase date:" with the value "2018-08-14". At the bottom, there are "Submit" and "Cancel" buttons.

• Please enter 2 or more characters

Title: S

Mileage: 42

Purchase date: 2018-08-14

Submit Cancel

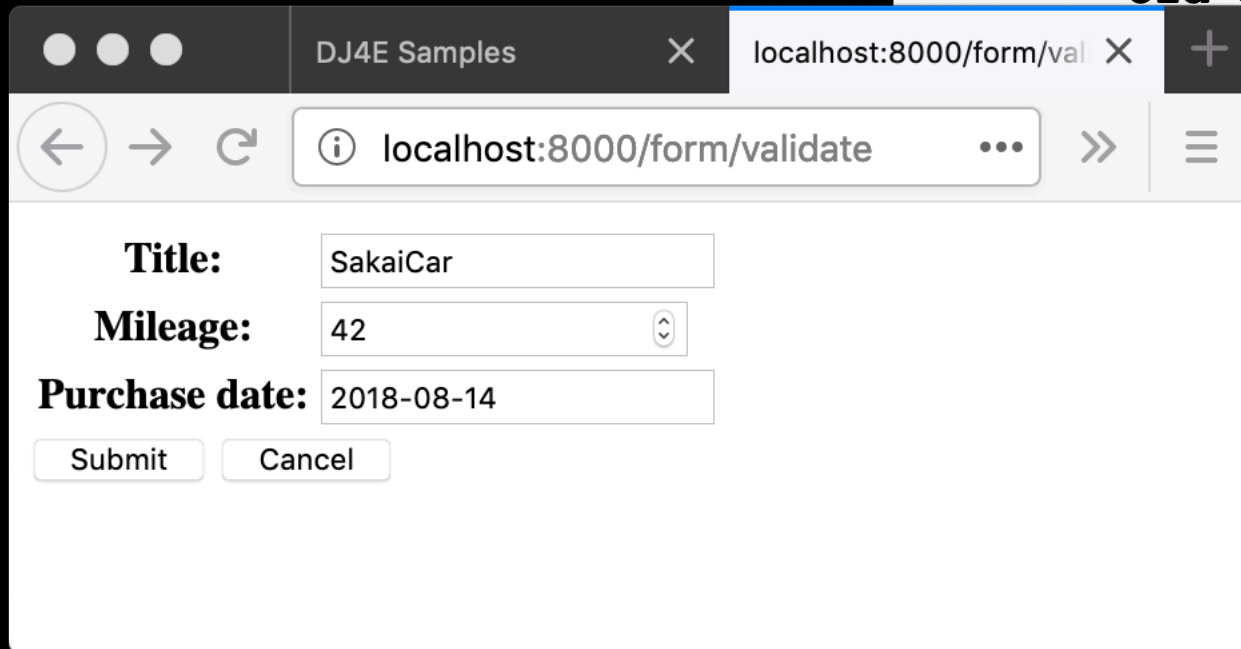
Django form validation

`dj4e-samples/form/forms.py`

```
class BasicForm(forms.Form):  
    title = forms.CharField(validators=[  
        validators.MinLengthValidator(2, "Please enter 2 or more characters")  
    ])  
    mileage = forms.IntegerField()  
    purchase_date = forms.DateField()
```

<https://docs.djangoproject.com/en/3.0/ref/validators/>

CST



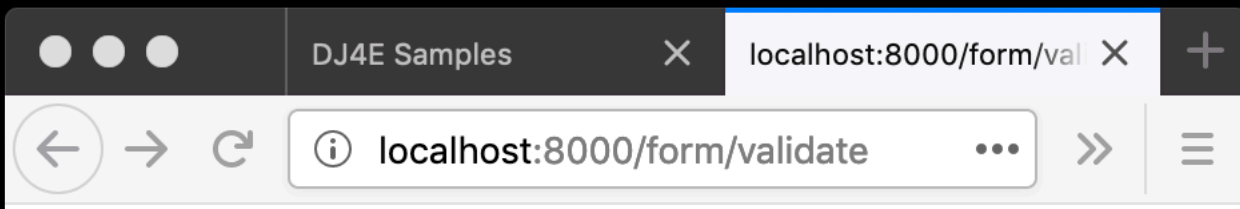
The screenshot shows a web browser window with the address bar displaying 'localhost:8000/form/validate'. The page contains a form with three input fields: 'Title' with the value 'SakaiCar', 'Mileage' with the value '42', and 'Purchase date' with the value '2018-08-14'. Below the fields are two buttons: 'Submit' and 'Cancel'.

```
class Validate(View):
    def get(self, request) :
        old_data = {
            'title': 'SakaiCar',
            'mileage' : 42,
            'purchase_date': '2018-08-14'
        }
        form = BasicForm(initial=old_data)
        ctx = {'form' : form}
        return render(request, 'form/form.html', ctx)

    def post(self, request) :
        form = BasicForm(request.POST)
        if not form.is_valid() :
            ctx = {'form' : form}
            return render(request, 'form/form.html', ctx)
        # Save the Data
        return redirect('/form/success')

def success(request) :
    return HttpResponse('Thank you!')
```

`dj4e-samples/form/views.py`



<https://samples.dj4e.com/form/validate>

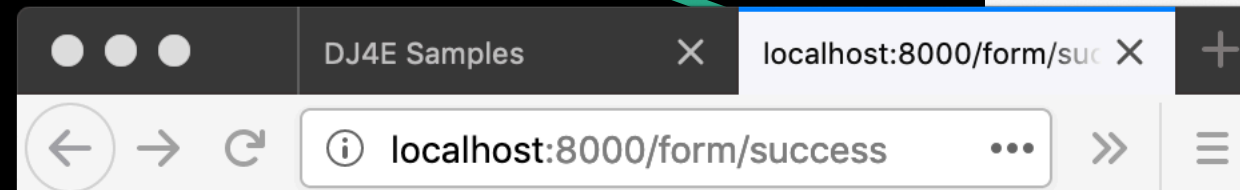
Title:

Mileage:

Purchase date:

POST

```
class Validate(DumpPostView):  
    def get(self, request):  
        old_data = {  
            'title': 'SakaiCar',  
            'mileage': 42,  
            'purchase_date': '2018-08-14'  
        }  
        form = BasicForm(initial=old_data)  
        ctx = {'form': form}  
        return render(request, 'form/form.html', ctx)
```



Thank you!

```
def post(self, request):  
    form = BasicForm(request.POST)  
    if not form.is_valid():  
        ctx = {'form': form}  
        return render(request, 'form/form.html', ctx)  
    If there are no errors, we would save the data  
    return redirect('/form/success')  
  
def success(self, request):  
    return HttpResponse('Thank you!')
```

DJ4E Samples × localhost:8000/form/val × +

localhost:8000/form/validate

Title:

Mileage:

Purchase date:

POST

```
class Validate(DumpPostView):
    def get(self, request):
        old_data = {
            'title': 'SakaiCar',
            'mileage': 42,
            'purchase_date': '2018-08-14'
        }
        form = BasicForm(initial=old_data)
        ctx = {'form': form}
        return render(request, 'form/form.html', ctx)
```

DJ4E Samples × localhost:8000/form/val × +

localhost:8000/form/validate

• Please enter 2 or more characters

Title:

Mileage:

Purchase date:

```
self, request):
    form = BasicForm(request.POST)
    if form.is_valid():
        ctx = {'form': form}
        return render(request, 'form/form.html', ctx)
    # If there are no errors, we would save the data
    return redirect('/form/success')

def post(self, request):
    return JsonResponse('Thank you!')
```

Models + Forms

Crossing the streams

<https://docs.djangoproject.com/en/3.0/topics/forms/modelforms/>

Form Structure is similar to Model Structure

`dj4e-samples/form/forms.py`

```
class BasicForm(forms.Form):
    title = forms.CharField(validators=[
        validators.MinLengthValidator(2, "...")])
    mileage = forms.IntegerField()
    purchase_date = forms.DateField()
```

`dj4e-samples/form/models.py`

```
class Cat(models.Model) :
    name = models.CharField(
        max_length=100,
        validators=[MinLengthValidator(2, "..")]
    )
    breed = models.CharField(max_length=100)
    comments = models.CharField(max_length=100, blank=True)
```


We can derive a form from a model

`dj4e-samples/form/models.py`

```
class Cat(models.Model) :
    name = models.CharField(
        max_length=100,
        validators=[MinLengthValidator(2, "..")]
    )
    breed = models.CharField(max_length=100)
    comments = models.CharField(max_length=100, blank=True)
```

`dj4e-samples/form/forms.py`

```
class CatForm(ModelForm):
    class Meta:
        model = Cat
        # fields = ['name', 'breed', 'comments']
        fields = '__all__'
```

dj4e-samples/form/views.py

```
class CatCreate(View):
    def get(self, request) :
        form = CatForm()
        ctx = {'form' : form}
        return render(request, 'form/form.html', ctx)

    def post(self, request) :
        form = CatForm(request.POST)
        if not form.is_valid() :
            ctx = {'form' : form}
            return render(request, 'form/form.html', ctx)

        # Save the form and get a model object
        newcat = form.save()
        x = reverse('form:main') + '#' + str(newcat.id)
        return redirect(x)
```

`dj4e-samples/form/views.py`

```
class CatUpdate(View):
    def get(self, request, pk) :
        oldcat = get_object_or_404(Cat, pk=pk)
        form = CatForm(instance=oldcat)
        ctx = { 'form': form }
        return render(request, 'form/form.html', ctx)

    def post(self, request, pk) :
        oldcat = get_object_or_404(Cat, pk=pk)
        form = CatForm(request.POST, instance=oldcat)
        if not form.is_valid() :
            ctx = { 'form' : form}
            return render(request, 'form/form.html', ctx)

        editcat = form.save()
        x = reverse('form:main')
        return redirect(x)
```

Summary

- Django forms
- Form Validation
- Models and Form

Acknowledgements / Contributions

These slides are Copyright 2019- Charles R. Severance (www.dr-chuck.com) as part of www.dj4e.com and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

Insert new Contributors and Translators here including names and dates

Continue new Contributors and Translators here

Additional Source Information

- Portions of the text of these slides is adapted from the text www.djangoproject.org web site. Those slides which use text from that site have a reference to the original text on that site. Django is licensed under the three-clause BSD license.